Pankesh et al., International Journal of Advanced Research in Computer Science and Software Engineering 2 (12), December - 2012, pp. 279-282

Volume 2, Issue 12, December 2012 **ISSN: 2277 128X**

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Image Encryption Using Pixel Shuffling

Pankesh Bamotra

SCSE, VIT University,
Vellore, India

Abstract— This paper deals with encryption of grayscale images using shuffling of sub-matrices of pixels. The idea is simple yet provides great security. The grayscale image to encrypt is converted into a matrix of grayscale values. Then depending upon level of encryption required the depth of encryption is chosen. Suitably the matrix is divided into sub-matrices which are shuffled in a random-order. This random order is preserved and serves as the shared secret between the two communicating parties and is transmitted on a secure channel using encryption techniques likes the RSA. On the receiver's end this sequence is retrieved and the sub-matrices are shuffled back to original positions followed by reconstruction of the original image.

Keywords— Cryptography, Encryption, Image processing, Pixel shuffling

I. INTRODUCTION

Image encryption is not at all an invention of modern times but a thing of ancient times. Polymaths like Leonardo Da Vinci have used the concept in their masterpieces of art. This paper deals with a simple image encryption technique using rearrangement or shuffling of the image pixels. The concept can be briefed in simple words. Initially we take a square grayscale image of size N × N. Next depending upon the level of encryption required we choose the depth of encryption required. In the implementation we took a 256 × 256 grayscale image and choose $\log_2 N$ or $\log_2 256 = 8$ as depth of encryption however choice of 6 or 7 would also work. This depth allows us to choose the order of sub-matrices which is equal to $N/2^{depth}$. The result is a total of $2^{2\times depth}$ sub-matrices. These sub-matrices are shuffled in a random manner and this random order serves as the shared secret between the parties. From the shuffled matrix of sub-matrices the encrypted image is constructed. This random order is sent over an encrypted channel to the receiving party along with the encrypted image. On the receiver's end the order of shuffling is retrieved along with the encrypted image and depth of encryption chosen. The encrypted image is again divided into sub-matrices of the order as calculated earlier and the original image is reconstructed by shuffling the sub-matrices back to their original positions.

II. METHODOLOGY

In the implementation part we begin by taking a 256 × 256 grayscale image, LENA.png. Using the numpy Python package we convert the grayscale image into a matrix of grayscale values i.e. all values in the matrix range between 0-255. Next we choose the depth of encryption which is visible in the encrypted image as shown later. After choosing the depth we divide the matrix into sub-matrices each of order equal to $256/2^{depth}$. In the result shown in the paper we choose depth = 8 so that we get sub-matrices, each of order equal to 2. These sub-matrices are then shuffled at random and the order of randomness is preserved which serves as the shared secret between the two communicating parties. This sequence alongwith the encrypted image and the order of sub-matrices is sent to the receiver. The message containing these three entities should be sent on an encrypted channel using public-key cryptographic algorithms like RSA. On the receiver's end the user retrieves these three entities and shuffles the sub-matrices back to their original positions to reconstruct the original image.

III. IMPLEMENTATION IN PYTHON

```
import numpy, math,matplotlib.pyplot as matplot
import random,  Image

image_name="lena512.bmp"
image_name_enc="lena_enc.bmp"

image_array = numpy.asfarray\
   (Image.open(image_name).convert('L'))

image_size=Image.open(image_name).size
print "Image size is :\t"+str(image_size)
max_depth=int(math.log(image_size[0],2))
```



```
def inp_depth(dep):
   return raw_input("Enter the depth "
                    "of encryption 1 - "+str(dep)+" :\t")

def matrix_stacking(input_matrix, div_factor):
   index=1
   matrix_3D_rec=[]
   temp=[input_matrix[0]]
   for i in xrange(div_factor):
      matrix_3D_rec.append([])
      for j in xrange(div_factor-1):
         matrix_3D_rec[i]=numpy.column_stack\
            ((temp[0],(input_matrix[index])))
         temp[0]=matrix_3D_rec[i]
         index+=1
      if index==no_o_div: break
      temp=[input_matrix[index]]
      index+=1
   temp=matrix_3D_rec[0]
   for i in xrange(1,div_factor):
      temp=numpy.row_stack((temp, matrix_3D_rec[i]))
   return temp

def matrix_split(image_array, no_o_div,div_factor):
   matrix_3D=[]
   index=0
   limiter=1
   row_increment=0
   col_increment=0
   for i in xrange(no_o_div):
      temp=limiter*div_factor-1
      matrix_3D.append([])
      if i<=temp:
         matrix_3D[index]=numpy.matrix\
            (image_array[row_increment:
         order_o_smatrix+row_increment,
         col_increment:order_o_smatrix+col_increment])
         col_increment+=order_o_smatrix
         index+=1
      else:
         limiter+=1
         row_increment+=order_o_smatrix
         col_increment=0
         matrix_3D[index]=numpy.matrix\
            (image_array[row_increment:
         order_o_smatrix+row_increment,
         col_increment:order_o_smatrix+col_increment])
         col_increment+=order_o_smatrix
         index+=1
   return matrix_3D

depth=int(inp_depth(max_depth))
while not(0<= depth <= max_depth):
   depth=int(inp_depth(max_depth))

div_factor=2**depth
no_o_div=div_factor**2
order_o_smatrix=image_size[0]/div_factor

print "Division factor : "+str(div_factor)
print "No. of divisions : "+str(no_o_div)
print "Order of sub-matrix : "+str(order_o_smatrix)
```



```
matrix_3D=matrix_split(image_array,no_o_div,div_factor)
#------------------------ENCRYPTION PART-------------------------#
range_list=[i for i in xrange(no_o_div)]                         #
random.shuffle(range_list)                                       #
matrix_3D = [ matrix_3D[i] for i in range_list]                  #
#-----------------------------------------------------------------#
#------------------------DECRYPTION PART-------------------------#
matrix_3D_original=[]                                            #
for i in xrange(len(range_list)):                                #
   matrix_3D_original.append([])                                 #
index=0                                                          #
for i in range_list:                                             #
   matrix_3D_original[i]=numpy.matrix(matrix_3D[index])          #
   index+=1                                                      #
#-----------------------------------------------------------------#
#------------------------DISPLAY PART----------------------------#
temp=matrix_stacking(matrix_3D, div_factor)                      #
matplot.gray()                                                   #
matplot.imshow(image_array)                                      #
matplot.figure()                                                 #
matplot.imshow(temp)                                             #
matplot.show()                                                   #
#-----------------------------------------------------------------#
```

IV. RESULTS

As a sample image a 256 × 256 size grayscale image LENA.png was taken and the results are shown in the figures below :-

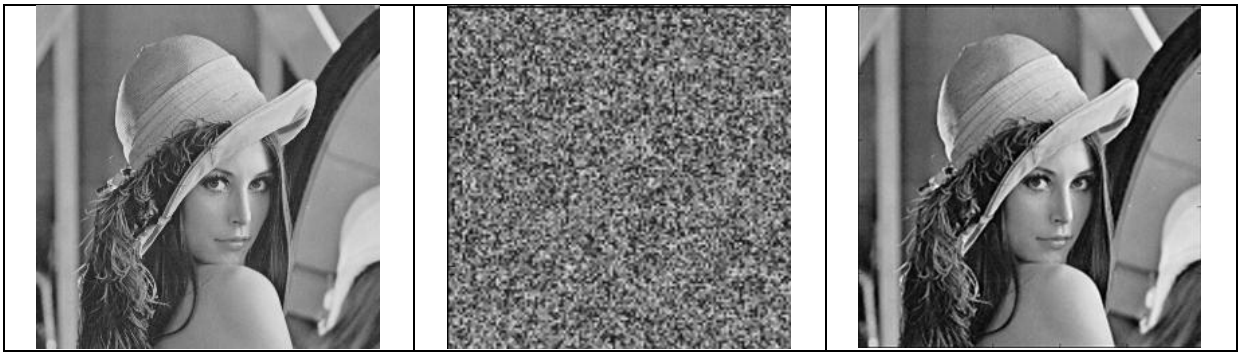

Fig.1ABCA. Original image* 256 × 256 B. Encrypted image* with depth=8 C. Decrypted image 256 × 256

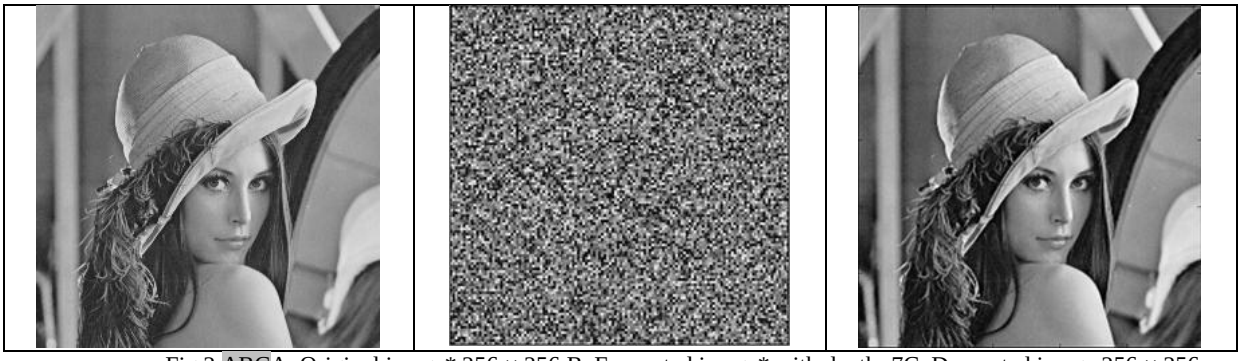

Fig.2 ABCA. Original image* 256 × 256 B. Encrypted image* with depth=7C. Decrypted image 256 × 256

*Scaled down to 50% of original size

V. CONCLUSIONS

It can be concluded from the above results that the image was properly encrypted and decrypted. The whole process took only 1.8180000782 seconds which clearly tells that the algorithm has been optimsed to work with large images too.

REFERENCES

[1] Sesha Pallavi Indrakanti and P.S.Avadhani, *Permutation based Image Encryption Technique,* International Journal of Computer Applications (0975 – 8887), Volume 28– No.8, August 2011.

[2] Ratinder Kaur and V. K. Banga, *Image Security using Encryption based Algorithm,* International Conference on Trends in Electrical, Electronics and Power Engineering (ICTEEP'2012) Jul 15-16, Singapore

[3] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, 2nd edition, Prentice Hall

[4] Shiguo Lian, *Multimedia Content Encryption*, 2009 edition, Auerbach Publication

